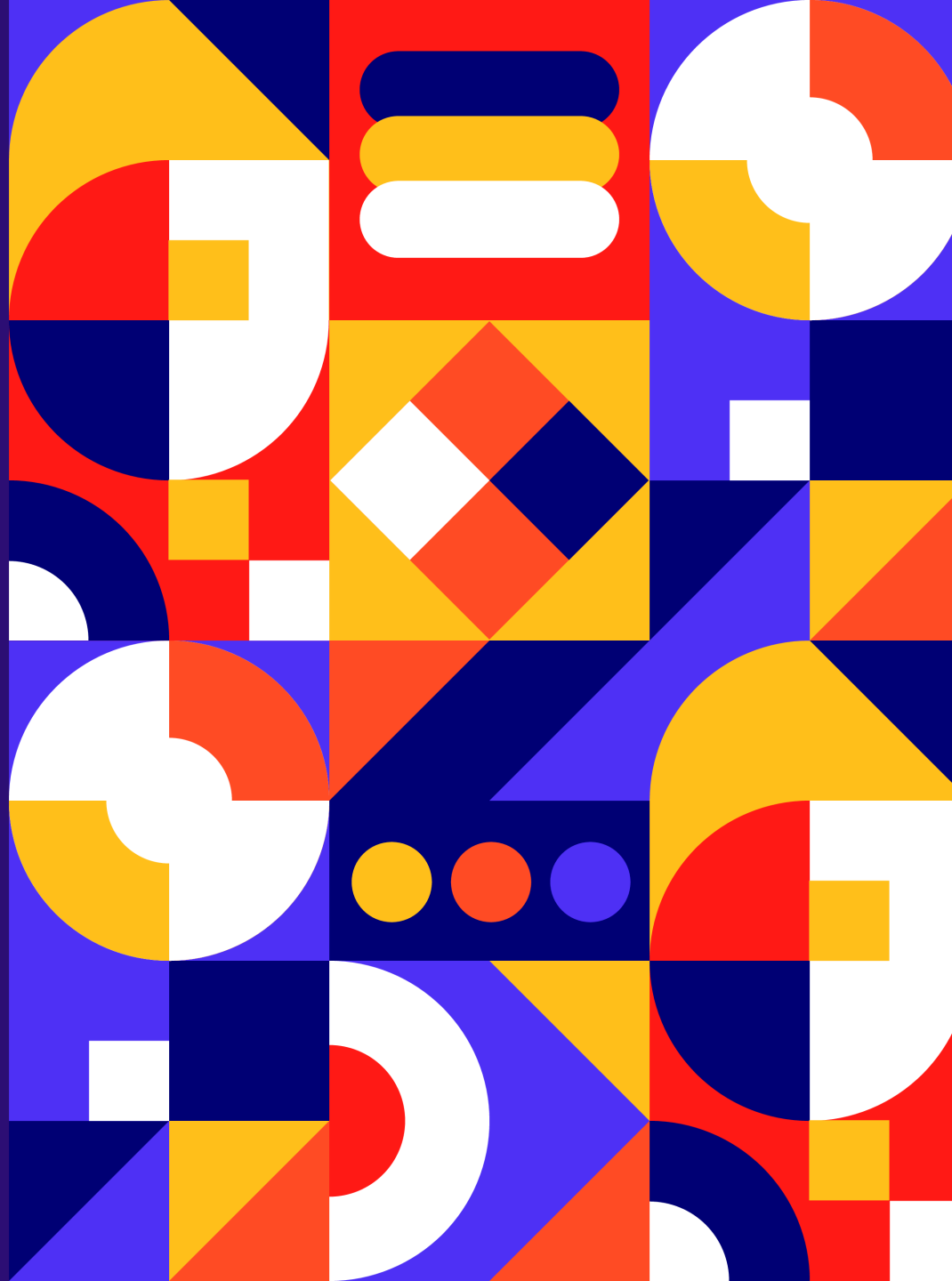
escola britânica de artes criativas & tecnologia

Módulo 26

Conhecendo o TypeScript

Gian Souza



TypeScript

O TypeScript é uma nova linguagem de programação, ela foi criada pela Microsoft no ano de 2012. Sua principal tarefa é adicionar a sintaxe de tipagem ao JavaScript, apesar de ser uma linguagem de programação o resultado do TypeScript será um código JavaScript, por isso recebe também o nome de superset.

Tipagem fraca

No JavaScript podemos começar uma variável string e depois, ao evoluir o programa, atribuir um número a essa variável, o que pode resultar em falhas quando o código for executado, com o TypeScript resolvemos esse problema, podemos dizer que uma variável receberá apenas números, todo o processo do TypeScript é feito em tempo de codificação e build (construção do projeto), se houver algum problema no código não teremos a geração do JS final.

Tipagem fraca

No JavaScript podemos começar uma variável string e depois, ao evoluir o programa, atribuir um número a essa variável, o que pode resultar em falhas quando o código for executado, com o TypeScript resolvemos esse problema, podemos dizer que uma variável receberá apenas números, todo o processo do TypeScript é feito em tempo de codificação e build (construção do projeto), se houver algum problema no código não teremos a geração do JS final.

Tipos básicos

No TypeScript possuímos alguns tipos básicos, primitivos, são ele: boolean, number, string, array, readonlyArray, tuplas, union e any.

Arrays: os arrays podem ser declarados através dos colchetes ou das palavras reservadas Array e readonlyArray:

```
const nomes: string[] = []
const nomes2: Array<string> = []
const nomes3: readonlyArray<string> = []
```

Tipos básicos

A única diferença que temos é ao utilizar o readonlyArray, onde os métodos para manipulação do array serão removidos, por exemplo, não poderemos adicionar um novo item com push.

No JavaScript podemos ter num mesmo vários tipos de dados, isso não é possível no TypeScript, onde temos que definir do que o array será composto, porém temos um tipo de dado que permite um comportamento parecido, a tupla.

Tipos básicos

Para criar uma tupla utilizamos os colchetes informando os tipos que serão aceitos:

```
const lista: [string, boolean, number] = [“ana”, true, 22];
```

Assim temos uma estrutura similar a de um array, também podemos informar um nome para cada índice, como se fossem campos:

```
const lista: [nome: string, estaEstudando: boolean, idade: number] = [“ana”, true, 22];
```

Tipos básicos

Apesar da forte tipagem que o TypeScript nos fornece, podemos aceitar mais um tipo em uma variável, utilizando o Union Type:

```
let idade: number | string;
idade = 20
idade = “20 anos”
```

Tipos básicos

O TypeScript fornece um tipo fraco, chamado any, do inglês: qualquer, onde podemos atribuir qualquer tipo de dado, este recurso deve ser usado em último caso:

```
let minhaVariavel: any;
minhaVariavel = [1, 2, 3, 4]
minhaVariavel = “gian”
minhaVariavel = {}
minhaVariavel = true
```

Tipos customizados

Além dos tipos básicos que o TypeScript possui, podemos criar nossos próprios tipos, considere:

```
const profissional = {
  nome: “Sabrina”,
  idade: 30,
  profissao: “designer”
}
```

A partir dessa estrutura já podemos criar um tipo: Profissional e toda variável do tipo Profissional deverá conter as propriedades definidas.

Tipos customizados

O tipo customizado ficaria:

```
type Profissional = {
  nome: string;
  idade: number;
  profissao: string;
}
const dev1: Profissional = {
  nome: “bruno”,
  idade: 27,
  profissional: “dev front-end”,
  salario: 5000 // não será válido pois essa propriedade não
faz parte do tipo Profissional
}
```

Tipagem em funções

As funções também possuem tipos, em seus argumentos e em seu retorno:

```
let somaDoisNumeros(n1: number, n2: number): number {
  return n1 + n2;
}
```

No exemplo acima estamos recebendo dois argumentos, ambos do tipo number e após o fechamento dos parênteses incluímos o tipo do retorno que também será um number.

Casting

Apesar da forte tipagem que o TypeScript nos fornece, podemos fazer com que um tipo seja entendido como outro tipo no compilador do TypeScript, por exemplo, podemos tratar um número como se fosse uma string.

Para isso utilizamos a palavra reservada as:

```
const nome: string = 10 as string;
```

O exemplo acima é uma má prática, afinal um nome não é um número, esse recurso deve ser utilizado com cautela e apenas em casos onde não conhecemos o tipo do dado que vamos receber, o que pode acontecer em integrações a APIs.

Orientação a objetos

A programação orientada a objetos no TypeScript também se faz presente, e ganhou recursos novos como o modificador de acesso protected, além disso uma mudança grande em relação ao JavaScript é que no TypeScript é necessário declarar as propriedades antes de inicializa-las no construtor.

Modificadores de acesso

Os modificadores de acesso são os responsáveis pelo encapsulamento de um membro em uma classe, no JavaScript possuímos os modificadores:

- público: padrão, acessível na classe, nas herdeiras e na instância;
- privado: acessível apenas na classe;
- estático: acessível na classe;

Modificadores de acesso

No TypeScript tivemos a introdução do modificador protegido, que já é presente em outras linguagens de programação, assim no TypeScript os modificadores são:

- público: padrão, acessível na classe, nas herdeiras e na instância;
- privado: acessível apenas na classe;
- estático: acessível na classe;
- protegido: acessível na classe e nas herdeiras;

O modificador privado é o mais restrito.

Interfaces

Uma outra nova funcionalidade que temos no TypeScript é a interface, ela já existe em linguagens de back-end, uma interface basicamente é um contrato, onde temos que seguir algumas regras para que seja satisfeito, por exemplo, um veículo, obrigatoriamente precisa ter a capacidade de acelerar e frear.

Interfaces

Na programação orientada a objetos só é permitido a herança de uma classe, não podemos herdar de mais de uma classe, porém com as interfaces podemos implementar várias.

Escrevemos uma interface assim:

```
interface IVeiculo {
  acelerar: () => boolean;
  frear: () => boolean;
  velocidadeMaxima: number;
}
```

Interfaces

Para implementar a interface IVeiculo, precisamos fazer a implementação dos métodos frear, acelerar e definir uma propriedade velocidadeMaxima que contenha um número.

```
class Carro implements IVeiculo {
  velocidadeMaxima = 120;
  acelerar() {..}
  frear() {..}
}
```

TypeScript e DOM

O TypeScript também pode ser utilizado para adicionar comportamentos ao sites, afinal o seu produto final é um arquivo JavaScript.

Por padrão a linguagem já possui alguns tipos pré-definidos, como HTMLElement quando utilizamos um querySelector, getElementById, etc.